Master in High Performance Computing

A performance study of Quantum ESPRESSO's diagonalization methods on cutting edge computer technology for high-performance computing

Supervisor(s):
Stefano de Gironcoli

Co-supervisors:
Ivan Girotto,
Filippo Spiga

Candidate:
Anoop Kaithalikunnel Chandran

3rd edition
2016–2017

This dissertation is dedicated to Dr V. Kanchana who motivated me to pursue my real passion

Acknowledgments

I would like to thank my advisor Prof. Stefano de Gironcoli for his guidance and encouragement. His support has been invaluable to me during the entire program. I also like to thank co-advisors of this project Ivan Girotto without his debugging skills this project would have taken ages and Filippo Spiga for initiating, introducing and pointing this project in the right direction.

I also appreciate the help that has been offered to me by Massimiliano Fatica and Josh Romero from NVIDIA. I'm also acknowledging the advice and help given by Dr Paolo Giannozzi and Dr Pietro Delugas. I would also like to thank my friends Emine Kucukbenli and Giangiacomo Sanna, for their advice and cheering discussions. I'm also extending my gratitude to all my MHPC batchmates and Alberto Sartori for their support.

The work presented in this thesis and the permanence in Trieste were supported by the Abdus Salam International Centre for Theoretical Physics (ICTP) and Quantum ESPRESSO Foundation (QEF). I'm grateful for the support of organizations such as CINECA, SISSA, CNR (in the person of Dr. Massimo Bernaschi) and MaX Centre of Excellence for the resources which were used in the research.

Anoop Chandran
2017-12-04
Master in High Performance Computing

Abstract

We explore the diagonalization methods used in the `PWscf` (Plane-Wave Self Consistent Field), a key component of the Quantum ESPRESSO open-source suite of codes for materials modelling. For the high performance of the iterative diagonalization solvers, two solutions are proposed. Projected Preconditioned Conjugate Gradient (PPCG) method as an alternative diagonalization solver and Porting of existing solvers to GPU systems using CUDA Fortran. Kernel loop directives (CUF kernels) have been extensively used for the implementation of Conjugate Gradient (CG) solver for general k-point calculations to have a single source code for both CPU and GPU implementations. The results of the PPCG solver for Γ-point calculation and the GPU version of CG have been carefully validated, and the performance of the code on several GPU systems have been compared with Intel multi-core (CPU only) systems. Both of these choices reduce the time to solution by a considerable factor for different input cases which are used for standard benchmarks using QE package

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

For more than four decades Density Functional Theory [2] and its applications have been a significant contributor in many fields of science as well as in industry. Due to its extensive applications, a number of computational packages for material simulation and modelling emerged. Among them, one of the prominent ones is Quantum ESPRESSO (QE). It is an integrated suite of computer codes for electronic-structure calculations and materials modelling [1, 3] written mainly in Fortran. This free, open-source package is released under the GNU General Public License (GPL) [4]. Its use has been extended outside the intended core research fields such as theoretical condensed matter and quantum chemistry to a vast community of users pursuing diverse research interests.

This software suite offers the possibility to introduce advanced programming techniques while maintaining highly efficient and robust code base. In the landscape of ever-evolving high-performance computing technology, this adaptive capability renders considerable advantages to QE software package. Even when it seemed the cutting edge software technology of Graphics Processing Unit's (GPU) for high-performance computing was out of reach for this Fortran code base, QE offered a mixed Fortran and CUDA C solution [5]. In 2017 NVIDIA offered its CUDA Fortran model analogues to CUDA C. This opened an abundance of opportunities for the code bases that are implemented in Fortran programming language including QE. Since QE is a modular suite which shares common libraries and data structures, a strategy of porting computationally heavy modules to GPU was possible. As part of the first implementation following the CUDA Fortran model, NVIDIA ported libraries such as FFT-3D and Davidson diagonalization with the help of a custom eigensolver [6] and released the first version of the QE-GPU package [7]. In this project, we attempt to extend this porting process to other vital parts of the QE code such as the Conjugate Gradient (CG) solver, Γ-point calculations, etc., This project also reflects on the performance analysis and comparison of the different diagonalization solvers on CPU and GPU systems.

Since iterative diagonalization is one of the most time-consuming procedures in the QE code, other methods have to be investigated and implemented. If one were to summarise the desired qualities of such a solver, then it should be at least as fast as Davidson and should consume less memory like CG. An ESLW project [8] was initialised with modularisation and investigation of iterative diagonalization as its objectives. As part of this Davidson and CG solver were extracted and implemented in such a way that other code bases can also make use of

it. In addition to that, a version of Davidson with the reverse communication interface (RCI) [9] and Projected Preconditioned Conjugate Gradient (PPCG) [10] methods were implemented. For the PPCG solver initial tests with Γ-point calculations suggest that it is an improvement over Davidson solver.

Chapter 4 and 5 are entirely devoted to the description of the computational methodology of this porting process and for the results and comparisons of the benchmarks. Chapter 3 deals with the CPU implementation details of the `PWscf` library of QE along with the discussions regarding the newly implemented PPCG solver.

CHAPTER 2

Theoretical Background

The fundamental postulates of quantum mechanics assert that microscopic systems are described by wave functions that characterize all the physical quantities of the system. For a solid system with a large number of microscopic subsystems (atoms) and having a huge number of particles, it is difficult to solve the Schrödinger equation $H\psi = E\psi$, analytically. For these solid systems the main interest is to find approximate solution of non-relativistic time independent Schrödinger equation. In general, Hamiltonian of such systems are defined by the kinetic energy of the electron and nuclei, electron-electron interaction, nuclei-nuclei interaction, electron-nuclei interaction. These interactions can be expressed in the following Hamiltonian (in atomic units $\hbar = |e| = m_e = \frac{1}{4\pi\epsilon_0} = 1$, E in the units of $27.21eV$, distance in the units of bohr radius),

$$H = -\sum_i \frac{1}{2}\nabla_i^2 - \sum_I \frac{1}{2M_I}\nabla_I^2 + \sum_{\substack{i,j \\ i\neq j}} \frac{1}{2}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} - \sum_{i,I} \frac{Z_I}{|\mathbf{R}_I - \mathbf{r}_i|} + \frac{1}{2}\sum_{\substack{I,J \\ I\neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{2.1}$$

The small indexes(i,j) are referring to electrons and capital (I,J) are for nuclei. This is known as many body Hamiltonian. Various number of approximations are adopted to solve this many body Hamiltonian, which are explained in detail below.

2.1 Approximations to solve Many-Body Problem

2.1.1 Born-Oppenheimer Approximation

Since nuclei are much heavier than electrons, their velocities are much smaller in comparison. To a good approximation, the Schrödinger equation can be separated into two parts: One part describes the electronic wavefunction for a fixed nuclear geometry. The second describes the nuclear wavefunction. This is the Born-Oppenheimer (BO) approximation. We assume that electrons move in an electrostatic field generated by the nuclei. Since we consider the nuclei to be at rest we can rewrite the Hamiltonian as

$$H = -\sum_i \frac{1}{2}\nabla_i^2 + \sum_{\substack{i,j \\ i\neq j}} \frac{1}{2}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} - \sum_{i,I} \frac{Z_I}{|\mathbf{R}_I - \mathbf{r}_i|} + \frac{1}{2}\sum_{\substack{I,J \\ I\neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{2.2}$$

In BO approximation, the total wave function is limited to one electronic surface, i.e. a particular electronic state. The BO approximation is usually very good, but breaks down when two (or more) electronic states are close in energy at particular nuclear geometries. Further approximations are needed to solve this Hamiltonian.

2.1.2 The Hartree approximation

For non interacting particles we can write the wave function as a product of individual particle wave function. Hartree considered each system obeying a Schrödinger equation and one can write the wave equation for an n particle system as,

$$\psi(\mathbf{r}_i) = C_N \prod_i^n \phi_i(\mathbf{r}_i) \tag{2.3}$$

Here interaction of one electron with the others are incorporated in an average way. We can write the Schrödinger equation as

$$\left(\frac{1}{2}\nabla^2 + V_{ext}(\mathbf{r}) + V_{additional}(\mathbf{r})\right)\phi_i = \epsilon_i \phi_i \tag{2.4}$$

Where $V_{ext}(\mathbf{r})$ is the average potential felt by the electron at $\mathbf{r}$ and $V_{additional}$ is related to electron-electron interaction. The limitations of this approximation include, not taking correlation into consideration, the wavefunction is not antisymmetric,etc,.

2.1.3 Hartree-Fock Method

One must include the antisymmetry as well in order to describe a wave function. Use of the slater determinant can be of use since it takes care of the spin. Interchanging the position of two electrons is equivalent to interchanging the corresponding column. If two electrons at the same spin interchange positions, ψ^D must change sign. This is known as exchange property and is the manipulation of Pauli principle. One of the most common ways of dealing with many - Fermion problem is to assume that each electron can be considered separately in the one electron approximation.

$$\psi^D(\mathbf{r}) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\mathbf{r}_1) & \chi_2(\mathbf{r}_1) & \dots & \chi_n(\mathbf{r}_1) \\ \chi_1(\mathbf{r}_2) & \chi_2(\mathbf{r}_2) & \dots & \chi_n(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(\mathbf{r}_n) & \chi_2(\mathbf{r}_n) & \dots & \chi_n(\mathbf{r}_n) \end{vmatrix} \tag{2.5}$$

$$H = \left(-\sum_i \frac{1}{2}\nabla_i^2 + V_{ext}(\mathbf{r}_i)\right) + \frac{1}{2}\sum_{\substack{i,j \\ i\neq j}} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{2.6}$$

Solving the Schrödinger equation with the slater determinant as the wave function, we will arrive at the equation given bellow:

$$\left(\frac{-1}{2}\nabla^2 + V_{ext} + \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}dr' - \frac{1}{2}\sum_{i,j,\sigma}\int \frac{\phi^*_{j,\sigma}(\mathbf{r}')\phi_{i,\sigma}(\mathbf{r}')\phi_{j,\sigma}(\mathbf{r})}{\phi_{i,\sigma}(\mathbf{r}')|\mathbf{r}-\mathbf{r}'|}drdr'\right)\phi_{i,\sigma}(\mathbf{r}) = \epsilon_i\phi_{i,\sigma}(\mathbf{r}) \tag{2.7}$$

This is called the Hartree-Fock equation. The Hartree-Fock equations describe non-interacting electrons under the influence of a mean field potential consisting of the classical Coulomb potential and a non-local exchange potential. The correlation energy accounts for the energy lowering due to quantum fluctuations. Since this is still a 3N dimensional problem, an analytical solution would be difficult. Even though this methods succeeded in describing various systems, it failed because of the poor exchange and correlation limits of electrons. Adopting further approximations were needed to minimise the problem. The triumph came with the formulation of Density Functional Theory (DFT) in solving this complex problem [11]. The same is explained in the following sections.

2.2 Density Functional Theory

Density Functional Theory (DFT) emanate from the Hohenberg-Kohn theory and Kohn-Sham equation [11]. This uses density as a fundamental quantity instead of wavefunction, which leads to a situation were the complexity of the problem can be effectively reduced from 3N variables to 3. The density of the electrons $\rho(\mathbf{r})$ can be expressed as,

$$\rho(\mathbf{r}) = N\int d^3\mathbf{r}_2 d^3 ... d^3\mathbf{r}_N \psi(\mathbf{r}, \mathbf{r}_2, ...\mathbf{r}_N)\psi^*(\mathbf{r}, \mathbf{r}2...\mathbf{r}_N) \tag{2.8}$$

The indication that the density can be used as a fundamental parameter just like the wavefunction was first suggested by L Thomas and E Fermi in 1927.

2.2.1 Thomas-Fermi-Dirac Equation

Thomas and Fermi independently considered the first three terms of the Hartree-Fock equation. At that time they were not aware of the exchange energy and neglected the correlation term. For a plane wave systems like homogeneous electron gas one can solve the HF equation and find out the approximate energy as follows. For a homogeneous electron gas $\phi_i(\mathbf{r}) = \frac{1}{\sqrt{V}}e^{i\vec{k}.\mathbf{r}}$ The exchange term for this model will be,

$$E_x = -\int \frac{3K_F}{4}\rho^{4/3}(\mathbf{r})dr \tag{2.9}$$

(2.10)

where $K_F = \left(\frac{3}{\pi}\right)^{1/3}$ Using this if we minimise the total energy by doing a variation with respect to density provided that the total electrons in the system remains the same. We get,

$$\frac{5C_k}{3}\rho(\mathbf{r})^{2/3} + V_{ext} + \frac{1}{2}\int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}dr' - \left(\frac{3\rho(\mathbf{r})}{\pi}\right)^{1/3} = \mu \tag{2.11}$$

The Lagrangian multiplier μ will be the chemical potential of the homogeneous electron gas. It shows that given a density it will give a number. The idea of using density as a fundamental variable is originated from this concept. But the density comes out by solving the Thomas-Fermi-Dirac equation wasn't accurate, in order to fix this Weizsäcker added the correction term $\frac{1}{8}\int \frac{|\nabla\rho(\mathbf{r})|^2}{\rho(\mathbf{r})}dr$ Thomas-Fermi becomes relevant when the system is dense or the kinetic energy is higher.

2.2.2 Hohenberg - Kohn Theorem

The question of treating $\rho(\mathbf{r})$ as a fundamental variable is answered by Hohenberg - Khon Theorem (HK). In 1964 Hohenberg and Kohn proved two theorems. The first theorem may be stated as follows: *for any system of interacting particles in an external potential $V_{ext}(\boldsymbol{r})$, the density uniquely determines the potential.* If this statement is true then it immediately follows that the electron density uniquely determines the Hamiltonian operator. The second theorem establishes a variational principle: *For any positive definite trial density $\rho_i(\boldsymbol{r})$,such that $\int \rho_i(\boldsymbol{r})dr = N$ then $E[\rho_i(\boldsymbol{r})] \geq E_o$*
HK theorem provides evidence for the one-to-one correspondence between external potential $V_{ext}(\mathbf{r})$ and ground state density $\rho_0(\mathbf{r})$. It gives good approximation to the ground density as well as the energy. But still we have to solve many electron Schrödinger equation. A practical implementation can be carried out using Kohn-Sham method which is discussed in the next section.

2.2.3 Khon-Sham Method

Since density is a fundamental variable we can write the variational principle in terms of the density functional:

$$E[\rho(\mathbf{r})] = T[\rho(\mathbf{r})] + V_{ext}[\rho(\mathbf{r})] + V_{ee}[\rho(\mathbf{r})] \tag{2.12}$$

Here $V_{ext}[\rho(\mathbf{r})] = \int V_{ext}\rho(\mathbf{r})dr$. But The kinetic and electron-electron functionals are unknown. If good approximations to these functionals could be found then a direct minimisation of the energy would be possible. Kohn and Sham proposed the following approach to approximating the kinetic and electron-electron functionals. They introduced a fictitious system of N non-interacting electrons [12] to be described by a single determinant wavefunction in N "orbitals" ϕ_i. In this system the kinetic energy and electron density are known exactly from the orbitals:

$$T_s[\rho(\mathbf{r})] = -\frac{1}{2}\sum_i^N \left\langle \phi_i \left| \nabla^2 \right| \phi_i \right\rangle \tag{2.13}$$

Here the suffix(s) emphasises that this is not the true kinetic energy but is that of a system of non-interacting electrons, which reproduce the true ground state density:

$$\rho(\mathbf{r}) = \sum_{i}^{N} |\phi_i^2| \tag{2.14}$$

The construction of the density explicitly from a set of orbitals ensures that it is legal – it can be constructed from an asymmetric wavefunction. If we also note that a significant component of the electron-electron interaction will be the classical Coulomb interaction or Hartree energy

$$V_H = \frac{1}{2} \int \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} dr dr' \tag{2.15}$$

The energy functional can be rearranged as:

$$E[\rho(\mathbf{r})] = T_s[\rho(\mathbf{r})] + V_{ext}[\rho(\mathbf{r})] + V_H[\rho(\mathbf{r})] + E_{XC}[\rho(\mathbf{r})] \tag{2.16}$$

Where, E_{XC} is called exchange correlation functional:

$$E_{XC}[\rho(\mathbf{r})] = T[\rho(\mathbf{r})] - T_s[\rho(\mathbf{r})] + V_{ee}[\rho(\mathbf{r})] - V_H[\rho(\mathbf{r})] \tag{2.17}$$

which is simply the sum of the error made in using a non-interacting kinetic energy and the error made in treating the electron-electron interaction classically. Applying variational theorem and minimising the energy with respect to $\rho(\mathbf{r})$ we get the equation (same as we did for Hartree-Fock equation):

$$\left[-\frac{1}{2}\nabla^2 + V_{ext} + \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} dr' + V_{XC}(\rho(\mathbf{r})) \right] \phi_i(\mathbf{r}) = \epsilon_i \phi_i(\mathbf{r}) \tag{2.18}$$

In which we have introduced a local multiplicative potential which is the functional derivative of the exchange correlation energy with respect to the density, $V_{XC} = \frac{\delta E_{XC}(\mathbf{r})}{\delta \rho(\mathbf{r})}$ This set of non-linear equations (the Kohn-Sham equations) describes the behaviour of non-interacting "electrons" in an effective local potential. For the exact functional, and thus exact local potential, the "orbitals" yield the exact ground state density. These Kohn-Sham equations have the same structure as the Hartree-Fock equations with the non-local exchange potential replaced by the local exchange-correlation potential V_{XC}.

2.3 A Numerical Approach

One could model the numerical approach in solving the Kohn-Sham equation as follows. The effective potential of the ground state of a system can be constructed provided an initial

guess for the density $\rho(\mathbf{r})$. The effective potential can be written as

$$V_{eff} = V_{ext} + \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} dr' + V_{XC} \tag{2.19}$$

Once V_{eff} is known, one could proceed to solve the Khon-Sham hamiltonian using the guess wavefunction to generate the new wavefunction. If the consistency between the new and the old wave functions are maintained after an iteration the system is said to have achieved self-consistency.

$$H_{KS} = -\frac{1}{2}\nabla^2 + V_{eff} \tag{2.20}$$

The initial guess is informed by the atomic positions, shape of the lattice, etc,. In this state solving Khon-Sham Hamiltonian means diagonalizing the Hamiltonian, i.e finding the eigenvalues and eigenvectors corresponding to the given Hamiltonian. This is essentially an iterative digitalization procedure. One could employ numerous methods to do this, depending on the resources available and intension. The output quantities of the numerical procedure can be used to determine the properties of the system such as total energy, forces, stress, etc,.

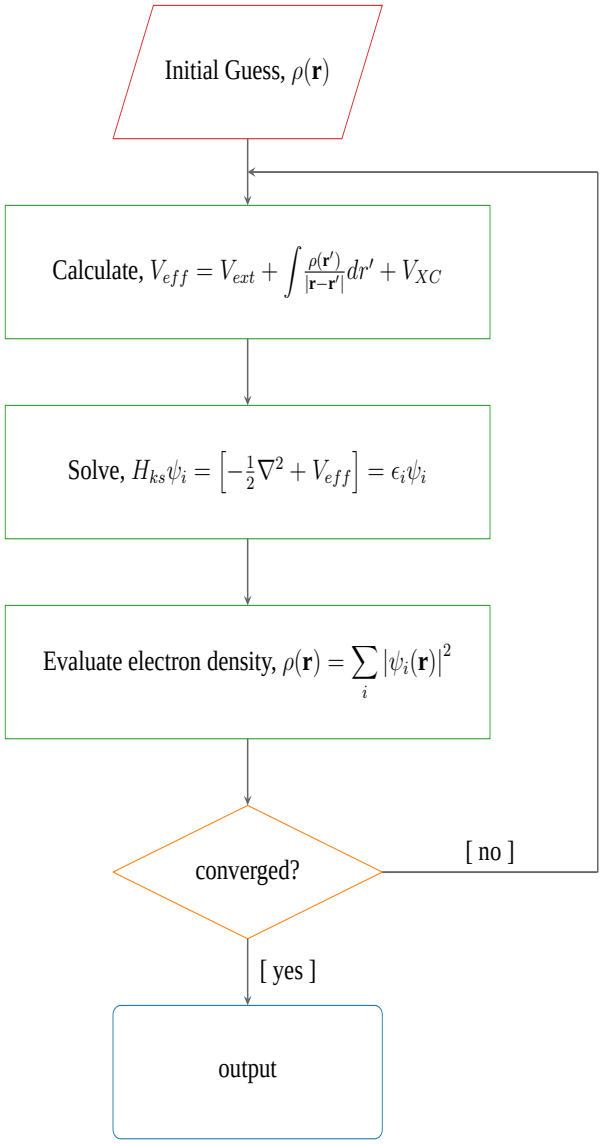


Figure 2.1: DFT Algorithm

CHAPTER 3

Implementation Details of Quantum ESPRESSO

QE is capable of performing many different kinds of calculations. The source code consists of approximately 520,000 lines of Fortran95 code, supplementary code in C, auxiliary scripts and Python. Two of the main libraries that are in the QE code base are `PWscf` (Plane-Wave Self-Consistent Field) and `CP` (Car-Parrinello). To make the treatment of infinite crystalline systems straightforward QE codes are constructed around the use of periodic boundary conditions and efficient convergence criteria. QE can be used for any crystal structure or supercell, and for metals as well as for insulators. The framework also includes many different exchange-correlation functionals such as LDA, GGA [13], advanced functionals like Hubbard U corrections and a few meta-GGA [14] and other hybrid functionals.

3.1 Structure of the PWscf code

For the purposes of this project we have used `PWscf` code which self-consistently solves Kohn-Sham(KS) equations described in the section 2.2.3.

In addition to solving KS orbitals and energies for isolated or extended/periodic systems, `PWscf` also performs the structural optimizations using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [15]. KS potential depends on the KS orbitals through density as described by the equation 2.19. This non-linear problem can be solved using an iterative procedure as described in the section 2.3 and the procedure is given in the Figure 2.1. A schematic representation of a typical execution of `PWscf` code involving structural optimization and self-consistent solution are given in the Figure 3.1. The self-consistency loop is an iteration over the density, until input and output densities are the same within a predefined threshold. The iterative diagonalization can be done using a block Davidson method or Conjugate Gradient (CG) method. This diagonalization is performed for each k-point. The number of occupied KS orbitals is determined by the number of electrons in the unit cell.

Majority of the time in the computational procedure is spent on the iterative diagonalization and calculation of density and the rest for initialization and post processing routines. To make the expensive parts efficient, QE suite offers parallel computing techniques targeting multi-core and distributed systems. This hybrid parallelism based on effective use of OpenMP, Message

Passing Interface (MPI), FFTW [16] and LAPAK [17] or ScaLAPACK [18] or ELPA [19] makes QE a high performance computing library.

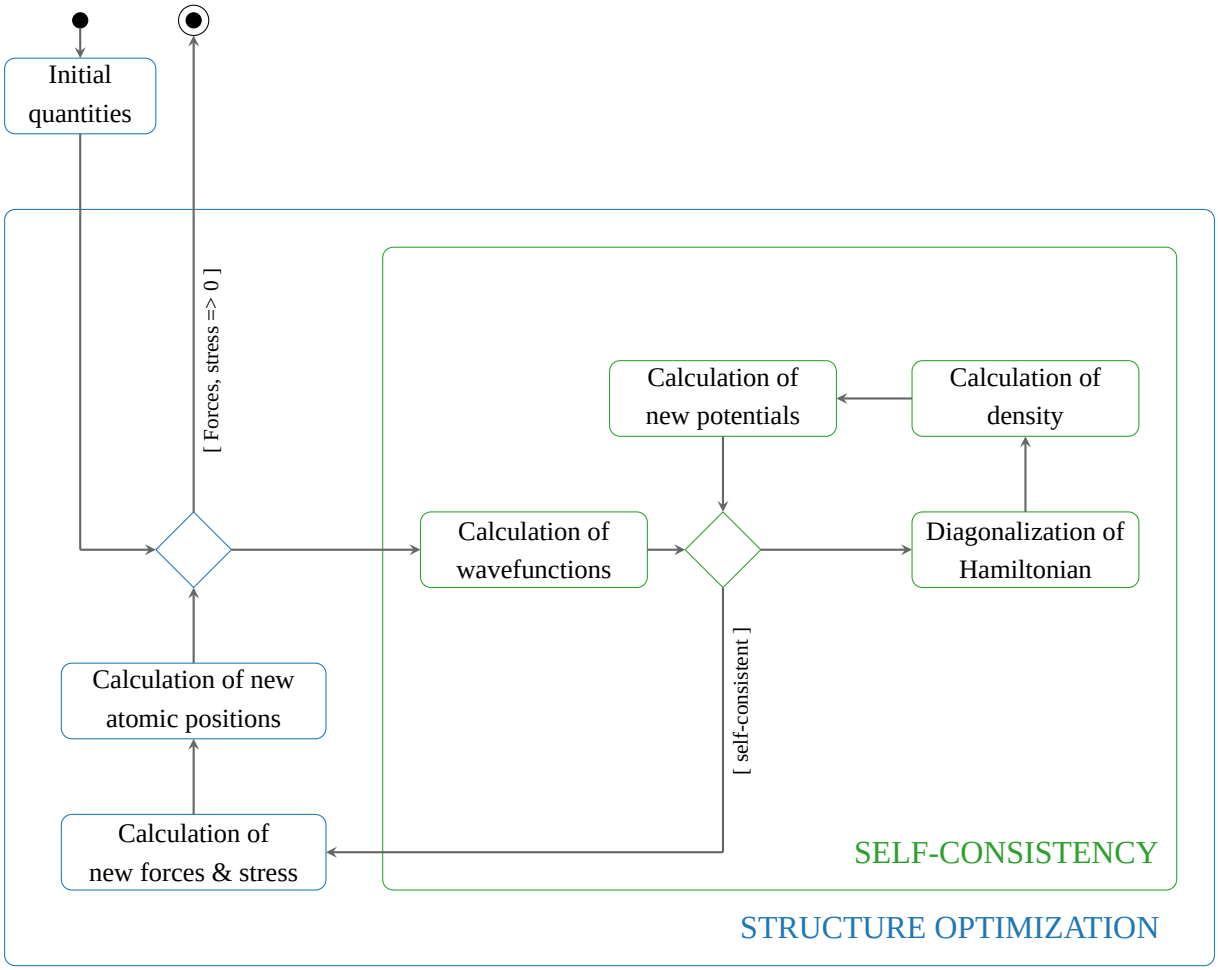


Figure 3.1: Schematic view of `PWscf` internal steps

The k-point parallelization which can be controlled using the run-time option `-npool` distributes the k-points into K pools. This allows for each pool having N_P MPI processes, where $N_P = \frac{N}{K}$, N being the total number of MPI processes. In the case of Davidson diagonalization an additional linear-algebra parallelization is possible using the option `-ndiag`. This option distributes the subspace diagonalization solution thereby making full use of the distributed linear solvers such as ScaLAPACK. A parallel distributed 3D-FFT is performed in order to transform physical quantities such as charge density and potentials between real and reciprocal space. Summary of parallelization levels in QE is given in the Table 3.1

3.2 Iterative diagonalization

As described earlier in this chapter the most time consuming portion of the `PWscf` code during self-consistency is the generalized eigenvalue solver therefore it is imperative that one should adopt a method which is efficient, resource friendly and fast. In QE two methods are implemented Davidson and Conjugate Gradient (CG). Depending on the need and resources

Group	Distributed quantities	Communications	Performance
Image	NEB images	Very low	Linear CPU scaling, Good load balancing; Does not distribute RAM
Pool	k-points	Low	Near-linear CPU scaling, Good load balancing; Does not distribute RAM
Plane-wave	Plane waves, G-vector, coefficients, R-space, FFT arrays	High	Good CPU scaling, Good load balancing, Distributes most RAM
Task	FFT on electron states	High	Improves load balancing
Linear algebra	Subspace Hamiltonians and constraints matrices	Very high	Improves scaling, Distributes more RAM

Table 3.1: Summary of parallelization levels in Quantum ESPRESSO [1]

available user can choose which algorithm to use to solve the following.

$$H\psi_i = \epsilon_i S\psi_i, \qquad i = 1, ..., N \tag{3.1}$$

N=Number of occupied states, S=Overlap matrix
Eigenvectors are normalized according to the generalized orthonormality constraints $< \psi_i|S|\psi_j >= \delta_{ij}$. The block Davidson is efficient in terms of number of $H|\psi >$ required but its memory intensive. CG on the other hand is memory friendly since bands are dealt with one at a time, but the need to orthogonalize to lower states makes it intrinsically sequential.

3.2.1 Davidson

Davidson starts with an initial set of orthonormalized trial orbitals $\psi_i^{(0)}$ and trial eigenvalues $\epsilon_i^{(0)} =< \psi_i^{(0)}|H|\psi_i^{(0)} >$. One can introduce the error on the trial solution ($g_i^{(0)}$) and the correction vectors ($\delta\psi_i^{(0)}$). The eigenvalue problem is then solved in the 2N-dimensional subspace spanned by the reduced basis set $\phi^{(0)}$ which is formed by $\phi_i^{(0)} = \psi_i^{(0)}$ and $\phi_{i+N}^{(0)} = \delta\psi_i^{(0)}$:

$$\sum_{k=1}^{2N} \left(H_{jk} - \epsilon_i S_{jk}\right) c_k^{(i)} = 0 \tag{3.2}$$

where $H_{jk} =< \psi_j^{(0)}|H|\psi_k^{(0)} >$ $\quad S_{jk} =< \psi_j^{(0)}|S|\psi_k^{(0)} >$ Conventional algorithms for matrix diagonalization are used in this step. A new set of trial eigenvectors and eigenvalues is obtained:

$$\psi_i^{(1)} = \sum_{j=1}^{2N} c_j^{(i)}\psi_j^{(0)}, \qquad \epsilon_i^{(1)} =< \psi_i^{(1)}|H|\psi_i^{(1)} > \tag{3.3}$$

and the procedure is iterated until a satisfactory convergence is achieved.

In the PWscf code the main routines that handle the Davidson are as follows:

- `regterg` , `cegterg`, which is an acronym of real/complex eigen iterative generalized
- `diaghg`, `cdiaghg`, which is an acronym of real/complex diagonalization H generalized
- `h_psi`, `s_psi`, `g_psi`, are code specific

3.2.2 Conjugate Gradient

The eigenvalue problem described in the equation 3.1 can be redefined as a minimization problem as follows:

$$min\left[<\psi_i|H|\psi_i> - \sum_{j\leq i}\lambda_j\left(<\psi_i|S|\psi_j>\right)\right] \tag{3.4}$$

where the λ_j are Lagrange multipliers. This can be solved with a preconditioned CG algorithm [20, 21]. The initial settings of solving this problem would involve assuming the first j eigenvectors have been calculated, where $j = i - 1$ and the initial guess being the i^{th} eigenvector. Such that $<\psi^{(0)}|S|\psi^{(0)}>= 1$ and $<\psi^{(0)}|S|\psi_j>= 0$. One can solve an equivalent problem by introducing a diagonal preconditioned matrix P and auxiliary function $y = P^{-1}\psi$.

$$min\left[<y|\tilde{H}|y> -\lambda\left(<y|\tilde{S}|y> -1\right)\right] \tag{3.5}$$

Where $\tilde{H} = PHP, \tilde{S} = PSP$, with additional orthonormality constraint $<y|PS|\psi_j>= 0$. By imposing the starting gradient $g^{(0)} = (\tilde{H} - \lambda\tilde{S})y^{(0)}$ is orthogonal to the starting vector one determines the value of $lambda$:

$$\lambda = \frac{<y^{(0)}|\tilde{S}\tilde{H}|y^{(0)}>}{<y^{(0)}|\tilde{S}^2|y^{(0)}>} \tag{3.6}$$

After imposing an explicit orthogonalization on $Pg^{(0)}$ to the ψ_j, the conjugate gradient $h^{(0)}$ is introduced with an initial value set to $g^{(0)}$ with a normalization direction $n^{(0)} = \frac{h^{(0)}}{<h^{(0)}|\tilde{S}|h^{(0)}>^{1/2}}$ The minimum of $<y^{(1)}|\tilde{H}|y^{(1)}>$ is computed along the direction $y^{(1)} = y^{(0)}\cos\theta + n^{(0)}\sin\theta$ as defined in [20]. Calculation of the minimum is performed yielding θ to be:

$$\theta = \frac{1}{2}atan\left(\frac{a^0}{\epsilon^{(0)} - b^{(0)}}\right) \tag{3.7}$$

Where $a^{(0)} = 2Re<y^{(0)}|\tilde{H}|n^{(0)}>, b^{(0)} =<n^{(0)}|\tilde{H}|n^{(0)}>$ and $\epsilon^{(0)} =<y^{(0)}|\tilde{H}|y^{(0)}>$
This procedure is iterated to calculate the conjugate gradient from the previous one using the Polak-Ribiere formula:

$$h^{(n)} = g^{(n)} + \gamma^{(n-1)}h^{(n-1)} \tag{3.8}$$

$$\gamma^{(n-1)} = \frac{<(g^{(n)} - g^{(n-1)})|\tilde{S}|g^n>}{g^{(n-1)}|\tilde{S}|g^{(n-1)}} \tag{3.9}$$

$h^{(n)}$ is subsequently re-orthogonalized to y^n.

In the PWscf code the main routines that handle the CG are as follows:

- rcgdiagg , ccgdiagg, which is an acronym of real/cmplx CG diagonalization generalize
- rotate_wfc_gamma, rotate_wfc_k, are for real/complex initial diagonalization
- h_1psi, s_1psi are code specific

3.3 Benchmarks

We are presenting a few CPU benchmarking results of QE version 6.1 done on a single computing node (20 cores) of Ulysses Cluster (SISSA) to give a perspective on the time to solution and scalability of the code. The MPI performance analysis of QE on one node (32 cores) of Broadwell (BWD), Knights Landing (KNL) and Skylake (SKL) architectures are done on Marconi Cluster (CINECA). The development of the GPU code is mainly done on the Drake machine provided by CNR. The main machines used in this project and their specifications are given in the table 3.2

Machines	Organization	Processors	Nodes	Cores	GPU's	GPU Version
Ulysses	SISSA	Intel Xeon E5-2680 v2 2.80 GHz	16	20	2	K20
Drake	CNR	Intel Xeon E5-2640 v3 2.60 GHz	1	16	4	K80
DAVIDE	CINECA	IBM Power8	45	16	4	P100
Marconi	CINECA	Intel Xeon E5-2697 v4 2.30 GHz	1512	54432	-	-

Table 3.2: Main machines used in this project

3.3.1 Details of the Test cases

Throughout this project mainly two test cases are used for benchmarking and analysing the PWscf code, both for CPU and GPU, the details of these test cases are given in table 3.3. The results correspond to calculation for a general k-point. One can improve the performance by doing a Γ-point calculation. This is not attempted here since the objective of this tests is that the results will be to used as a comparison with GPU performance in later chapters. The GPU performance analysis are done for a general k-point.

3.3.2 PWscf time to solution on CPU

Time to solution of the PWscf code for the test cases of SiO2 and AUSURF112 for a single node (Ulysses) is given in the table 3.4. This single node performance will be later compared

Properties	SiO2	AUSURF112
Lattice parameter (alat)	$22.5237 a.u.$	$38.7583 a.u.$
Number of atoms/cell	109	112
Number of atomic types	2	1
Number of electrons	582.00	1232.00
Number of k-points	1	2
Kohn-Sham states	800	800
Number of plane waves	16663	44316
Kinetic energy cutoff	$20 Ry$	$15 Ry$
Charge density cutoff	$80 Ry$	$100 Ry$
Convergence threshold	1.0×10^{-09}	1.0×10^{-06}
Mixing beta	0.7000	0.7000

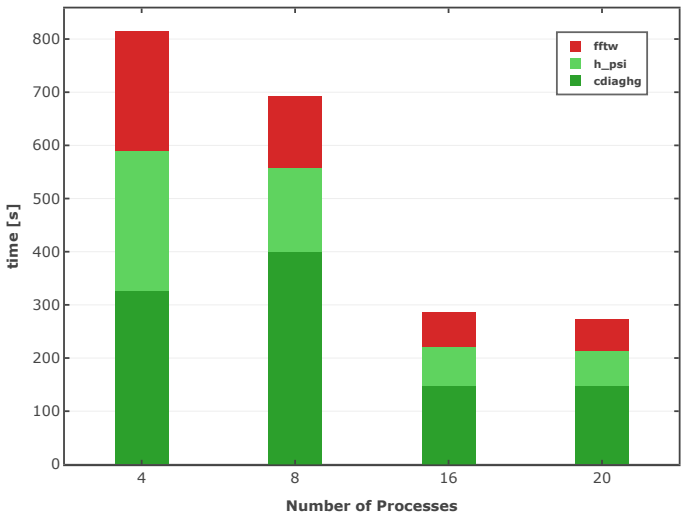
Table 3.3: Details of the test cases used for benchmarking

to the GPU performance. All the CPU versions are compiled with `Intel` compilers and used `mkl` from Intel

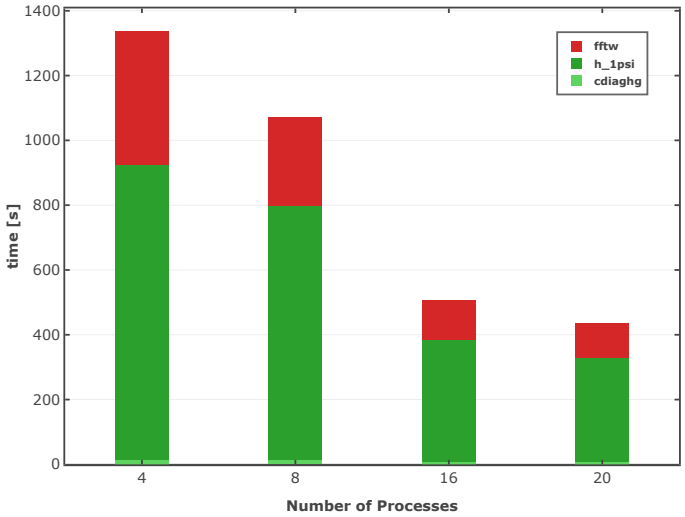
Test Case	Algorithm	4	8	16	20
SiO2	David	1218.68	900.00	405.68	370.84
SiO2	CG	2834.70	2295.16	1098.01	1009.51
AUSURF112	David	4740.00	3385.19	1870.86	1646.10
AUSURF112	CG	18300.00	11880.00	10380.00	10800.00

Table 3.4: `PWscf` Time to solution in seconds on a single node of Ulysses (SISSA)

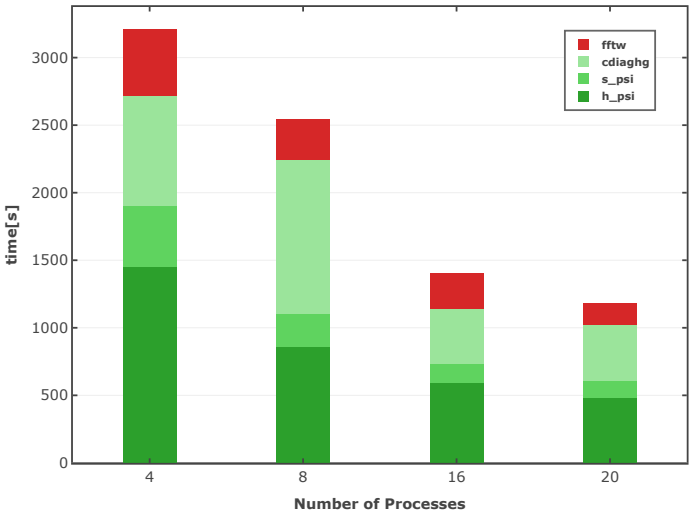
The main subroutines as described in the section 3.2.1 and 3.2.2 for the Davidson and CG methods behave differently in terms of speed and scaling. The Objective of the following graphs is to analyse the behaviour of Davidson and CG methods with varying problem sizes. A detailed benchmark on the CPU performance for multiple nodes is reported in the paper [1, 3].



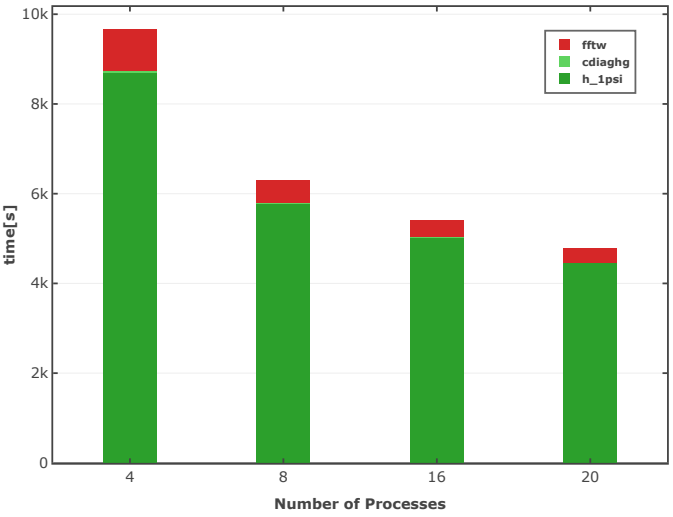
(a) SiO2-PWscf with Davidson



(b) SiO2-PWscf with CG



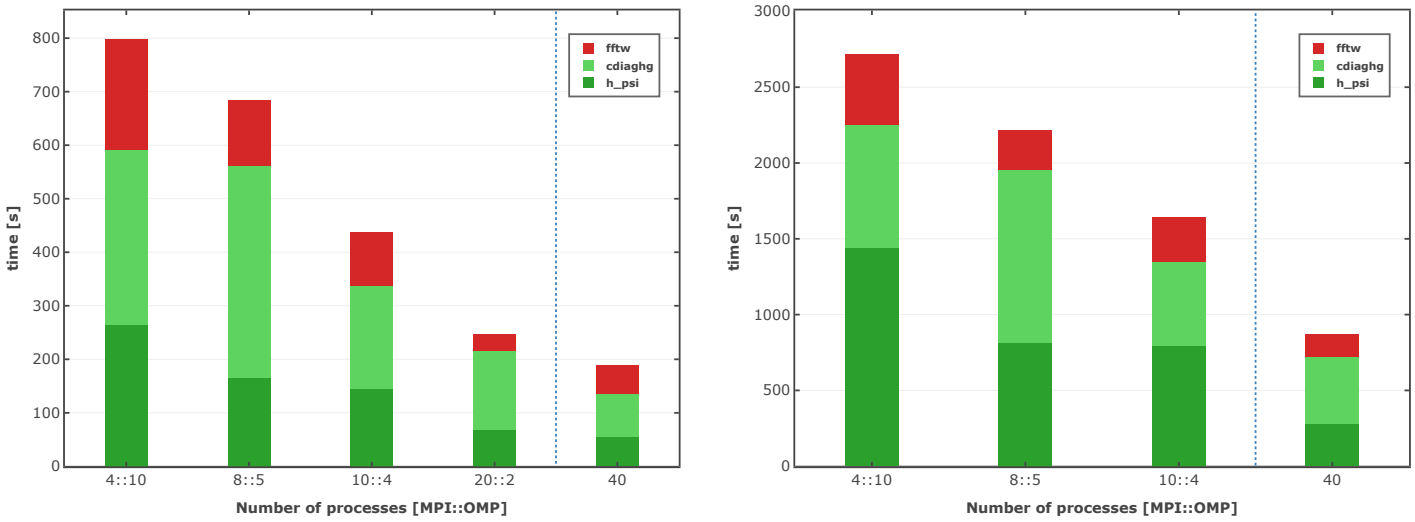
(c) AUSURF112-PWscf with Davidson



(d) AUSURF112-PWscf with CG

Figure 3.2: MPI only performance of Davidson and CG on a single node of Ulysses

An observation that can be made immediately from table 3.4 and the Figure 3.2 is that the Davidson method requires less time to achieve convergence compared to CG . This behaviour was expected. On analysing the `cegterg` subroutine performance one can see that for Davidson the scaling depend heavily on the problem size contrary to `cdiaghg` subroutine of CG. However the massive difference in the time to solution for these methods cannot be disregarded. `FFT` library maintains the same pattern in both cases.

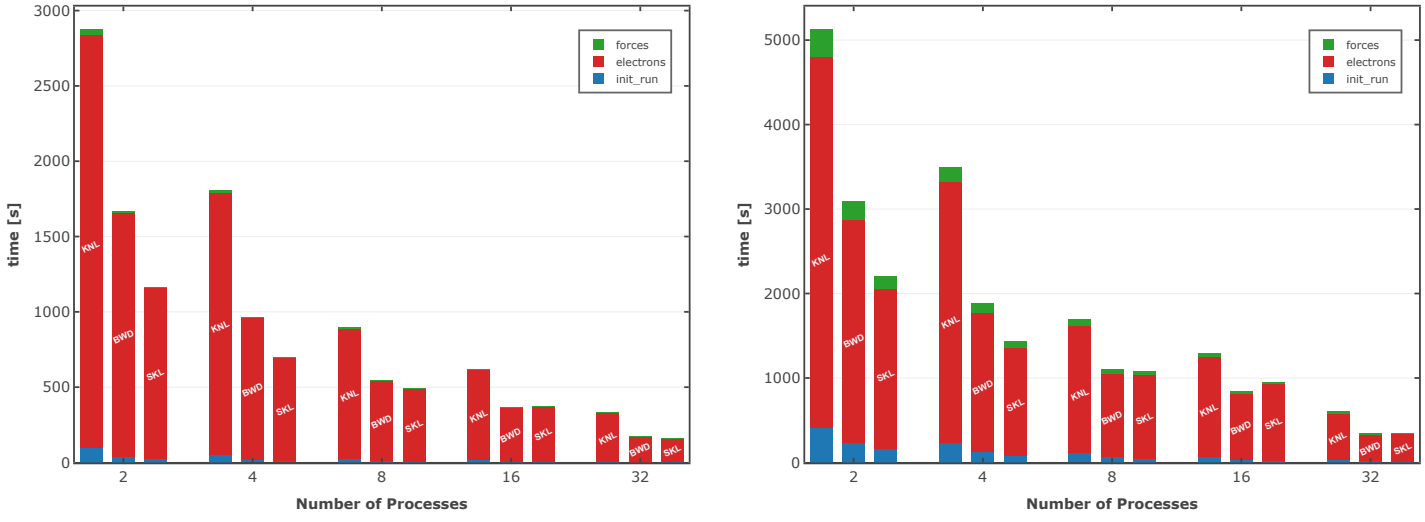


(a) SiO2–PWscf with Davidson (b) AUSURF112-PWscf with Davidson

Figure 3.3: MPI with OPENMP performance of Davidson on two nodes of Ulysses

The OpenMP implementation in PWscf is far from being efficient as shown in the figure 5.5. In an ideal case all the time to solutions should be the same as the MPI only performance on the two nodes(on the right of vertical line (blue) in figure 5.5).

The MPI performance of PWscf using Davidson on one node (32 cores) of Marconi cluster using Broadwell (BWD), Knights Landing (KNL) and Skylake (SKL) architectures are given in the figure 3.4. Total time to solution is the sum of time taken by `init_run`, `electrons` and `forces`. Among the three architectures BWD and SKL offers a higher performance compared to KNL.



(a) PWscf with Davidson for SiO2 (b) PWscf with Davidson for AUSURF112

Figure 3.4: MPI only performance of PWscf on KNL, BWD and SKL architectures of Marconi

3.3.3 Limitations and Possible Improvements

Even though Davidson is the preferred method of diagonalzation in the QE community due to its small time to solution compared to CG, it is memory intensive. It requires a work space of at least $(1 + 3 * david) * nbnd * npwx$ Where $david$ is a number whose default is set to 4 and can be reduced to 2, $nbnd$ is the number of bands and $npwx$ is the number of plane-waves. By construction Davidson diagonalization is faster but not so forgiving on memory. On the other hand since CG deals with bands one at a time it consumes less memory than Davidson but as a side effect the time to solution is much higher since it looses the additional parallelization level which is available in Davidson. This difference becomes apparent in GPU systems since GPU devices have a limited amount of memory.

3.4 Overview of Alternative Diagonalization Methods

Due to the factors mentioned in section 3.3.3 there have been a number of attempts made to find an alternative method which can combine the strength's of both Davidson and CG i.e. Fast and less memory consuming. Main possible candidates under consideration are Projected Preconditioned Conjugate Gradient (PPCG) algorithm [10] and Parallel Orbital-updating (ParO) algorithm [22]. During the Electronic Structure Library Workshop 2017 conducted by e-cam [23] at International Centre for Theoretical Physics (ICTP), a project has been initialized by Prof. Stefano de Gironcoli (SISSA) as an attempt to completely disentangle the iterative eigensolvers from the `PWscf` code. This project is hosted in the e-cam gitlab server under the url `https://gitlab.e-cam2020.eu/esl/ESLW_Drivers`. The aim of this ESLW project is to develop an independent eigensolver library by extracting the solvers from `PWscf` and extending them such a way that other code bases can also make use of the same. As a result of this ESLW project, Davidson, CG methods are detached from `PWscf` and development of other versions of Davidson which make use of Reverse Communication Interface (RCI) paradigm [9] and diagonalization methods such as PPCG [10] and ParO [22] are initiated. Even though the development of these methods are in its infancy they are showing promising results.

3.4.1 Projected Preconditioned Conjugate Gradient (PPCG)

PPCG method follows the algorithm given in Algorithm 1. Which is essentially an amalgamation of Davidson an CG. Each band or a small group of bands are updated by diagonalizing a small $(3 \times blocksize) \times (3 \times blocksize)$ matrix. This matrix is built from the current X, the orthogonal residual and the orthogonal conjugate direction. Since the bands are dealt with few at a time this method is memory efficient at the same time it is not as slow as CG, since its not dealing with one band at a time. This additional parallelization offered by this method in terms of group of bands which can be treated independently, would make this method as fast as Davidson. Comparison between Davidson and PPCG on a small test case for a gamma point calculation is given in the figure 3.6.

Algorithm 1: PPCG

subroutine PPCG (A, T, $X^{(0)}$,X)

INPUT : A - The matrix, T - a preconditioner, $X^{(0)}$ - a starting guess if the invariant subspace $X^{(0)} \in \mathbb{C}^{n,k}$ associated with k smallest eigenvalues of A

OUTPUT: X - An approximate invariant subspace $X \in \mathbb{C}^{n,k}$ associated with smallest eigenvalues of A

$X \longleftarrow orth(X^{(0)})$; $P \longleftarrow [\,]$

while *convergence not reached* **do**

  $W \longleftarrow T(AX - X(X * AX))$

  $W \longleftarrow (I - XX^*)W$

  $P \longleftarrow (I - XX^*)P$

  for *j=1,...,k* **do**

    $S \longleftarrow [x_j, w_j, p_j]$

    Find the smallest eigenpair(θ_{min}, c_{min}) of $S^*ASc = \theta S^*Sc$, where $c^*S^*Sc = 1$

    $\alpha_j \longleftarrow c_{min}(1), \beta_j \longleftarrow c_{min}(2)$; and $\gamma_j \longleftarrow c_{min}(3)$($\gamma_j = 0$ at the initial step)

    $p_j \longleftarrow \beta_j w_j + \gamma_j p_j$

    $x_j \longleftarrow \alpha_j x_j + p_j$

  end

  $X \longleftarrow orth(X)$

  if needed perform the Reyleigh-Ritz procedure within span(X)

end

end subroutine PPCG

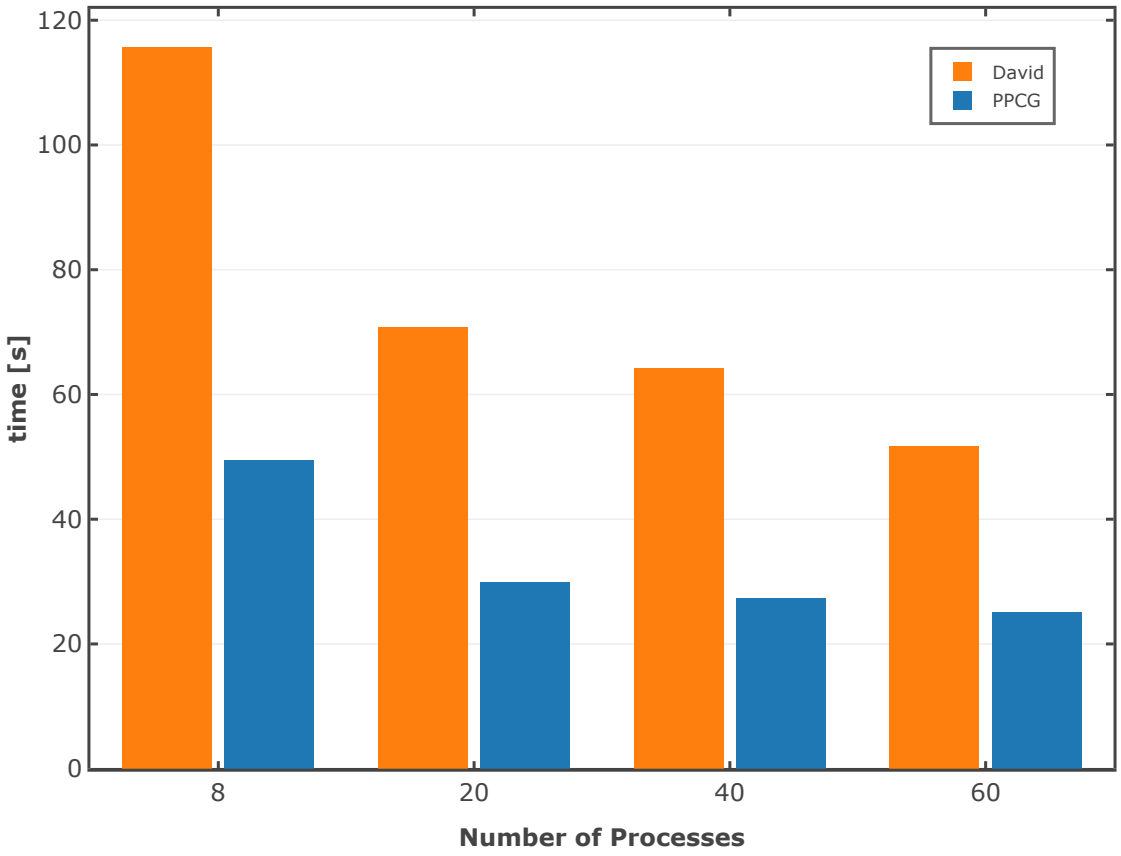


Figure 3.5: Comparison between Davidson and PPCG algorithms for a problem size of 512 for MPI only calculation on Ulysses cluster

This simple test might suggest that the PPCG is better than Davidson. However this test was done on a small problem size. Since Davidson algorithm performs best with large sizes, further analysis has to be made before making concrete argument in favour of PPCG. Presently the Γ-point calculation is implemented in PPCG. Future development of this ESLW project would include extending the current implementation for general k-point calculations and optimizing the code.

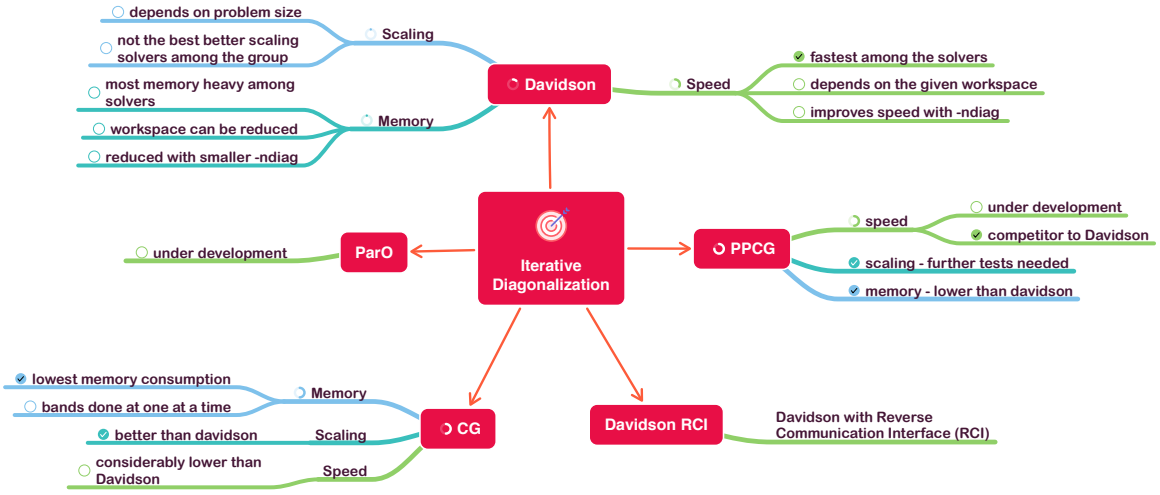


Figure 3.6: ESLW project-Current methods under development

CHAPTER 4

Work-flow and description of the implementation

With the release of a freely distributed PGI compiler in October 2016 (version 16.10) came an abundant number of possibilities in scientific software products. There are a large number of code bases which uses Fortran. The choices of running them on the GPU was limited before CUDA-Fortran which is supported by PGI. The alternate solutions include converting the code to a C compatible version and use the CUDA C programming model. For years such a solution existed for QE implemented and maintained by Filippo Spiga [5,24]. This was written in CUDA C and bundled with the original Fortran source code. This kind of solutions has their drawbacks, the main one being the maintainability of the code since there will be two versions of the code. Therefore after the last mirror release of QE version 5 this mixed code base was discontinued. The advantage of CUDA Fortran is that it can enable the GPU capability without drastic changes in the original source code. CUDA Fortran is implemented in the PGI compilers. A new version compatible with QE version 6 has been developed on CUDA Fortran [7] with all significant computations carried out on GPUs, with an added advantage of having one code base for CPU and GPU. The CUDA Fortran enabled QE is available for download at `https://github.com/fspiga/qe-gpu`

4.1 CUDA Programming Model and CUDA Fortran

CUDA-enabled GPU's are capable of running tens of thousands of threads concurrently. This functionality is possible due to a large number of processor cores available on each GPU device. These processor cores are grouped into multiprocessors. The design of the programming model is done such that it can utilise the massive parallelism capability of the GPU's. A subroutine run on a device is called a kernel, and it is launched with a grid of threads grouped into thread blocks. These thread blocks can operate independently of one another thereby allowing scalability of a program. In addition to that data can be shared between these threads enabling a more granular data parallelism. The available multiprocessors can schedule these block of treads in any order allowing the program to be executed on any device with any number of multiprocessors. This scheduling is performed automatically so that one can concentrate on the partition of the program into factions that can be run and parallelised on a block of treads.

CUDA programming model allows utilising both CPU and GPU to perform computations. This hybrid computing capability can be used to achieve massive performance improvements. A typical sequence of operations for a simple CUDA Fortran code is:

- Declare and allocate host (CPU) and device (GPU) memory
- Initialize host data
- Transfer data from the host to the device
- Execute one or more kernels which perform operations on the device data
- Transfer results from the device to the host

There are a number of good programming practices to be observed when moving data back and forth from host to device and vice-versa. Among those the most noticeable being the bandwidth of PCI's and bandwidth between device's memory and GPU, where the former is an order of magnitude lower than that of the later. From a performance point of view, the less number of data transfers between the host and the device the better will be the performance.

4.1.1 CUDA Fortran specifications

The `allocate()` command and the assignment operator's are overloaded in the CUDA Fortran model so that it can also allow allocation of the device memory (using `device` attribute) and data transfer between the host and device memory spaces. The Fortran2003 source allocation construct, `allocate(A, source=B)`, for cloning `A` to `B` is also extended. In CUDA Fortran, if the `A` array is defined as a device array and `B` is defined as host array, then contents of `B` will be copied over the PCI bus to `A`. These methods of data transfer are all blocking transfers, in that control is not returned to the CPU thread until the transfer is complete. This prevents the possibility of overlapping data transfers with computation on both the host and device. For concurrent computation one might employ CUDA API function `cudaMemcpyAsync()` to perform asynchronous data transfers

Kernels, or subroutines are typically invoked in host code just as any subroutine is called, but since the kernel code is executed by many threads in parallel an additional `execution configuration` has to be provided to indicate thread specifications, which typically done as follows:

```
! my_function is the kernel or subroutine
call my_function <<< N, M >>> (argument_list)
```

Where, `N` = Number of thread blocks, `M` = Number of threads per thread block.
The information between the triple chevrons is the `execution configuration` which dictates how many device threads execute the kernel in parallel. Data elements are mapped to threads using the automatically defined variables `threadIdx`, `blockIdx`, `blockDim`, and `gridDim`. Thread blocks can be arranged as such in a grid and threads in a thread block can be arranged in a multidimensional manner to accommodate the multidimensional nature of the underlying problem. Therefore both N and M are defined using a derive type `dim3` which has x,y and z components.

One of the functionalities provided by CUDA Fortran is that it can automatically generate and invoke kernel code from a region of host code containing tightly nested loops. Such type of kernels are referred to as CUF kernel's. A simple example of a CUF kernel is:

```
!$cuf kernel do <<<*,*>>>
do i=1, n
 ! a_d is a device array, b is a host scalar
 a_d(i) = a_d(i) + b
end do
```

In the above code wild-cards are used to make the runtime system determine the execution configuration. Even though the scalar `b` is a host variable it is passed as a kernel argument by value upon the execution of the CUF kernel.

4.1.2 Porting strategy

Quantum ESPRESSO has a large community of users, so it would be preferable to change the code in such a way that it doesn't impact the existing structure of the code. CUDA Fortran is designed to make this task easy. Especially the use of CUF kernel's which allow porting of the code without modifying the contents of the loops. This functionality is extensively used in porting the CPU code to GPU systems. This can be illustrated using a simple example.

In the following code subroutine `add_scalar` is making use of the module `array_def` which defines the array `a(n)` to make a simple addition with a scalar.

```
module array_def
 real :: a(n)
end module array_def
!
! Following subroutine make use of the module array_def
subroutine add_scalar
 use array_def, only: a
 do i=1, n
  a(i) = a(i) + b
 end do
end subroutine add_scalar
```

Converting this code to a GPU friendly one can be done as follows:

```
module array_def
 real :: a(n)
 real,device :: a_d(n)
end module array_def
!
subroutine add_scalar
#ifdef USE_GPU
 use array_def, only: a => a_d
#else
 use array_def, only: a
#endif
!$cuf kernel do <<<*,*>>>
 do i=1, n
  a(i) = a(i) + b
 end do
end subroutine add_scalar
```

When compiling without the preprocessor directive `USE_GPU`. The code will behave the same as the CPU version. Ignoring the CUF kernel execution configuration. When GPU is enabled the device allocated array `a_d(n)` is used instead, for a calculation respecting the CUF kernel. If the arrays used in the subroutine are explicitly passed then one can make use of the `device` attribute to achieve the same effect as above. The following code illustrate the same.

```
subroutine add_scalar(a,n)
 real:: a(n)
#ifdef USE_GPU
 attributes(device) :: a
#endif
!$cuf kernel do <<<*,*>>>
 do i=1, n
  a(i) = a(i) + b
 end do
end subroutine add_scalar
```

Since most of the subroutines in Quantum ESPRESSO are of the above type. In some unavoidable cases the code is redefined for GPU inside the preprocessor directive statements.

4.1.3 Libraries Used, Modified Subroutines and Files

The porting of performance critical libraries in QE version 6 using CUDA Fortran was initiated by NVIDIA. Main parts of the code that was ported include forward and inverse 3D FFTs, Davidson iterative diagonalization using a dense eigensolver [6] for general k-point calculations, computation of the forces etc,. In this project an attempt has been made to port the conjugate gradient (CG) algorithm of QE for general k-point(complex dtype). This is a necessary step towards porting the complete code to GPU and to study the advantageous of CG for a QE-GPU version.

In the source code of QE, the main subroutine which handles the CG algorithm for general k-point is `ccgdiagg()` as discussed in the section 3.2.2 which is in the `ccgdiagg.f90` file. This subroutine is called from `diag_bands_k` subroutine in the `c_bands.f90` file. The `rotate_wfc_k` subroutine required for the CG method is already implemented for the davidson method. All the calculations done in the `ccgdiagg` are analogous to the CPU implementation. As an alternative to LAPACK/ScaLAPACK libraries `Cublas` library is used for the GPU portions of the code. Two of the main `cublas` subroutines that are used for the CG method are `cublasDdot` and `cublasZgemv`. `cublasDdot` for calculating the dot product between two vectors and `cublasZgemv` for matrix vector operation. Other functions include `h_1psi` and `s_1psi` which applies the Hamiltonian and the S matrix to a vector psi, these are also ported to GPU as part of this project. `mp.f90` file is changed for including `mp_sum_cv_d`, the mp_sum of complex vector type. Most of the tightly nested loops are handled with the help of CUF kernels.

In the present stage of the porting process the major computational routines for general k-point calculation are implemented. Γ-point calculation routines are currently under development.

CHAPTER 5

Results and performance

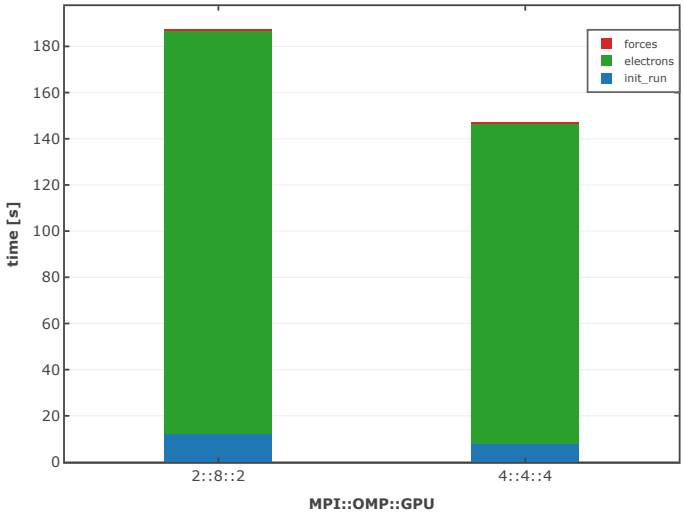
We devote the following sections to discuss several results that were obtained studying the behaviour of the test cases given in the table 3.3 on NVIDIA Tesla K80 GPU cards known as Kepler on the Drake machine and Tesla P100 GPU known as Pascal on the DAVIDE cluster. Which mainly include the performance of Davidson and CG methods on GPU's with regards to their speed and memory consumption and a comparison between the GPU and CPU speeds.

5.1 Performance results

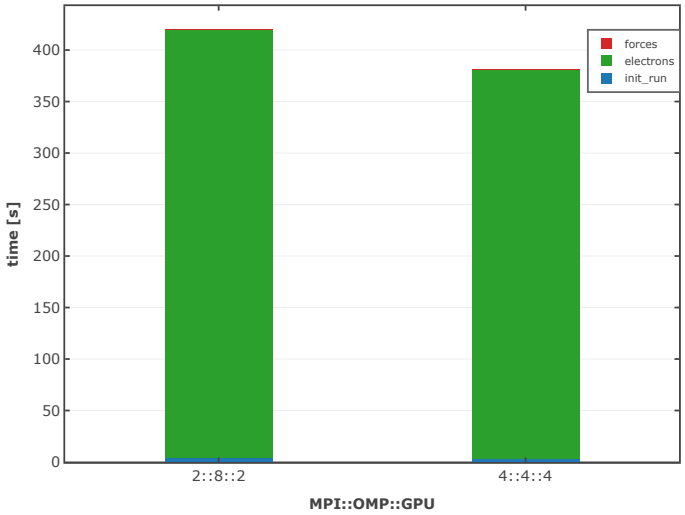
The results of the tests on a single node of the Drake machine with K80 GPU's are given in the figure 5.1. All tests are performed by binding one process per core. Time to solution is considerably lower than the CPU counterpart of the respective test cases given in section 3.3.2. Even though it wouldn't be fair to consider the scaling with only two data points, one is tempted to say that the CG code maintains its better scalability feature on GPU as seen from the CPU results. This feature is slightly prominent in the case of multiple node performances as given in the figure 5.2 done on P100 GPU's for AUSURF112. It can be said that for this particular test case Davidson already has achieved saturation. But Davidson method clearly maintains an advantage in terms of speed. It is also interesting to note that the time to solution decreases at least twice in magnitude when going from K80 to P100. This behaviour can be attributed to a faster memory subsystem and the higher double precision performance of P100 cards of nearly 5 TeraFLOPS, for K80 GPU's its 3 TeraFLOPS.

By observing the comparison on the memory usage of these two methods given in the figure 5.3, one could say that what CG loses in speed it gains in memory. This behaviour was expected since it is the same that we have seen in CPU as well. Davidson by construction consumes more memory than CG. This dissimilarity becomes apparent when using large tests since the GPU devices have limited memory. For a K80 device this limitation is 12GB, and for a P100 it is 16GB. As shown in the same figure the memory advantage of CG is at least half that of Davidson when using the default `-ndiag` when running the `PWscf`. This heavy memory usage can be brought down for Davidson by setting `-ndiag` to the minimum that is 2, which will reduce the workspace allowed for diagonalization. This modification, however, would also bring down the performance of Davidson diagonalization linearly. In the figure 5.3 the horizontal bar (green colour-online) on the Davidson indicate memory consumption when the `-ndiag` is set to

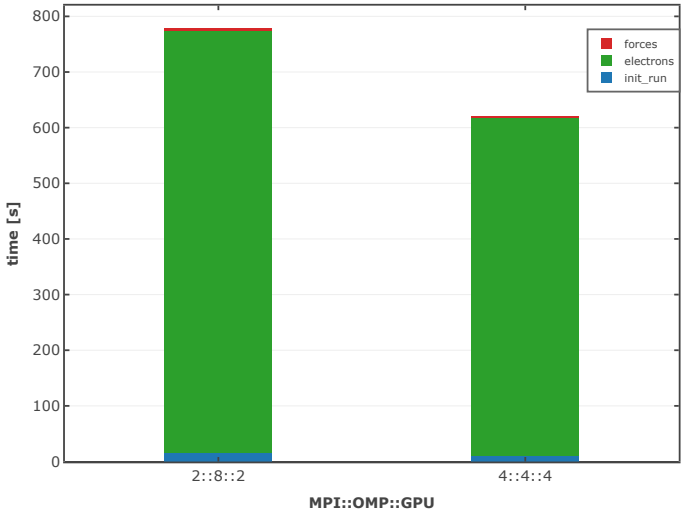
minimum, even in this configuration CG conserves the memory advantage.



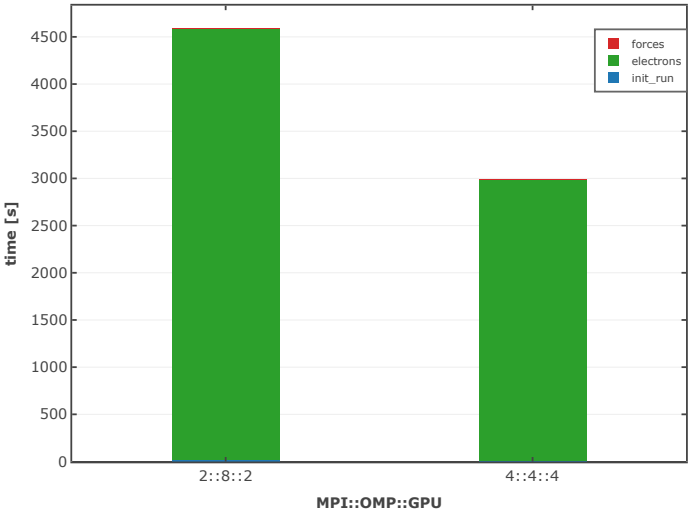
(a) SiO2-PWscf with Davidson



(b) SiO2-PWscf with CG

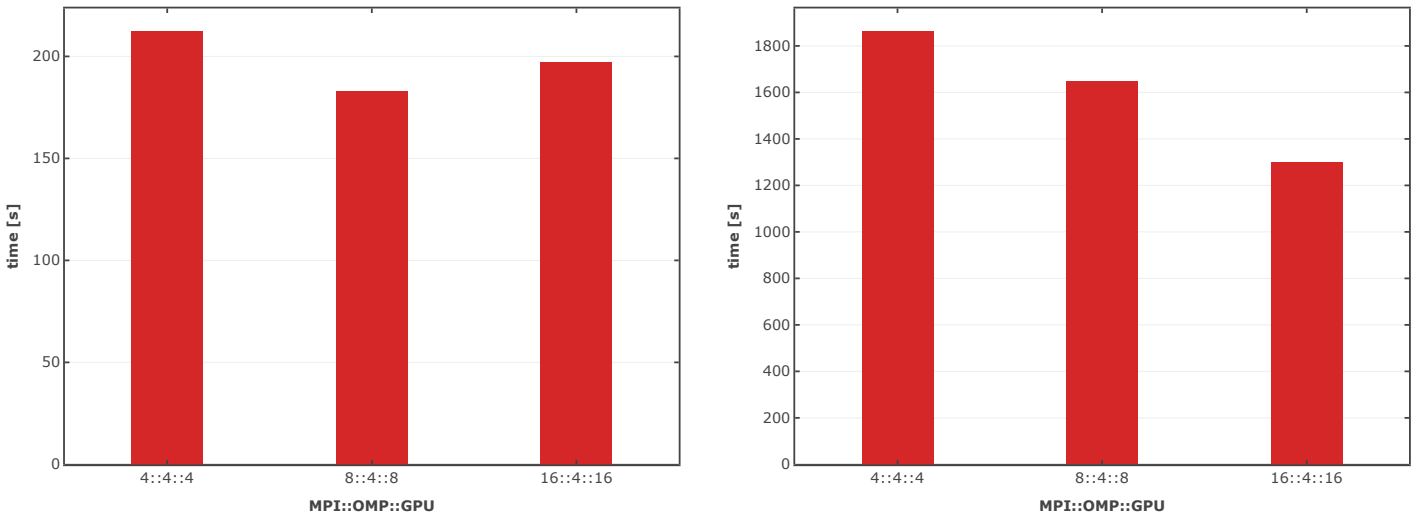


(c) AUSURF112-PWscf with Davidson



(d) AUSURF112-PWscf with CG

Figure 5.1: GPU performance of Davidson and CG on a single node of Drake



(a) `PWscf` with Davidson for AUSURF112 (b) `PWscf` with CG for AUSURF112

Figure 5.2: 1,2 and 4 node performance of Davidson and CG on DAVIDE

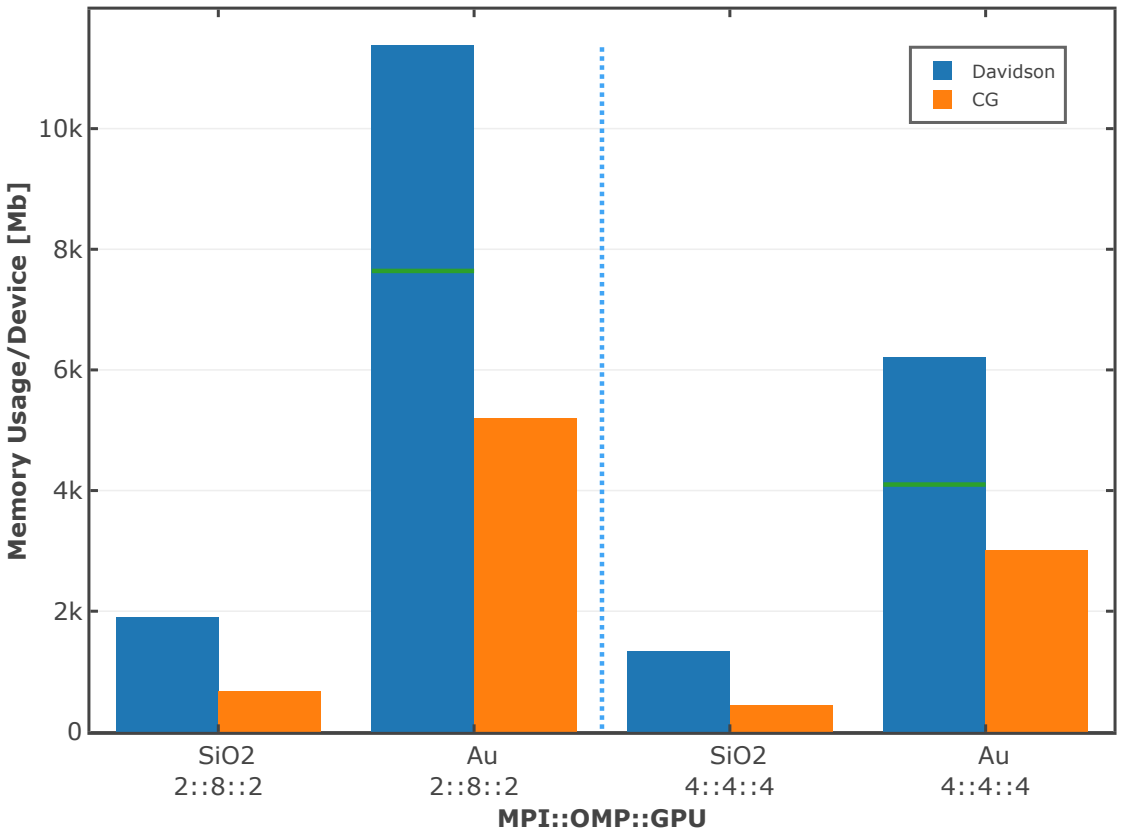


Figure 5.3: Memory consumption per GPU device

Figure 5.4 shows the average of the volatile GPU utilisation as reported from the `nvidia-smi` tool when the kernels are utilising the GPU. This infers that the implementation of the CG method uses GPU more efficiently than the Davidson in the tested cases.

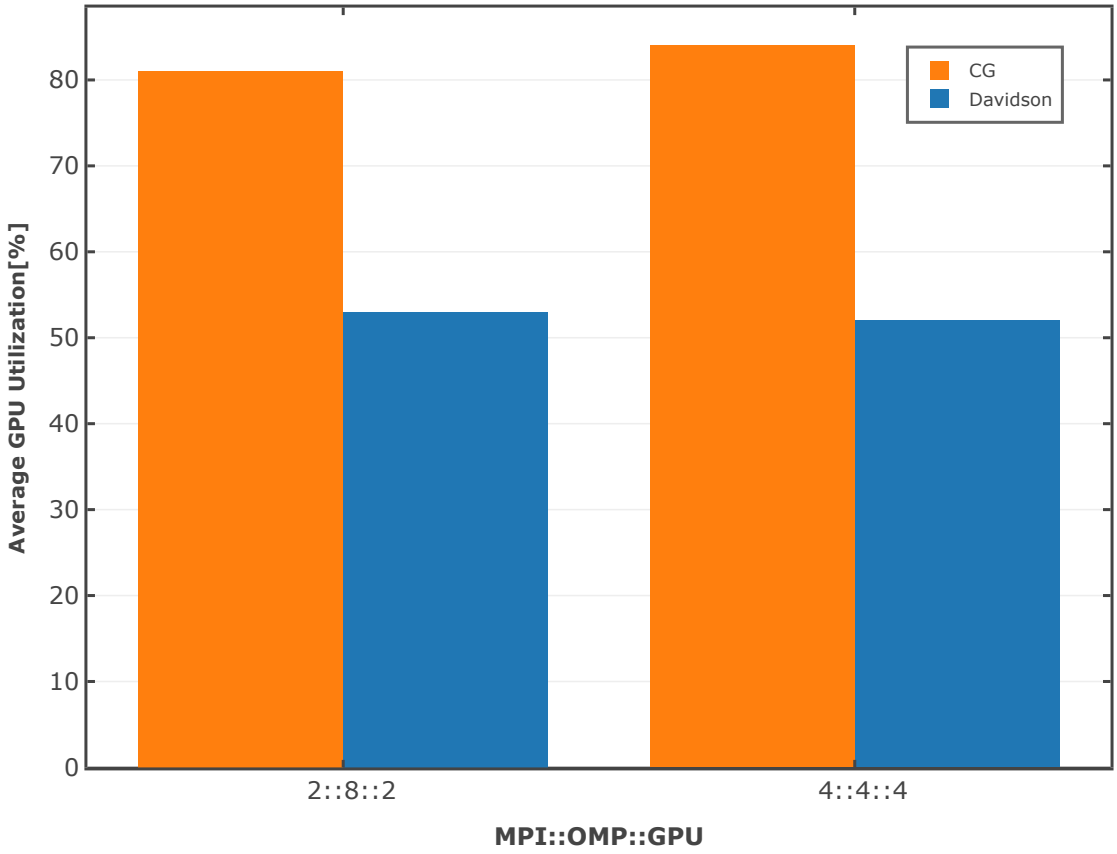
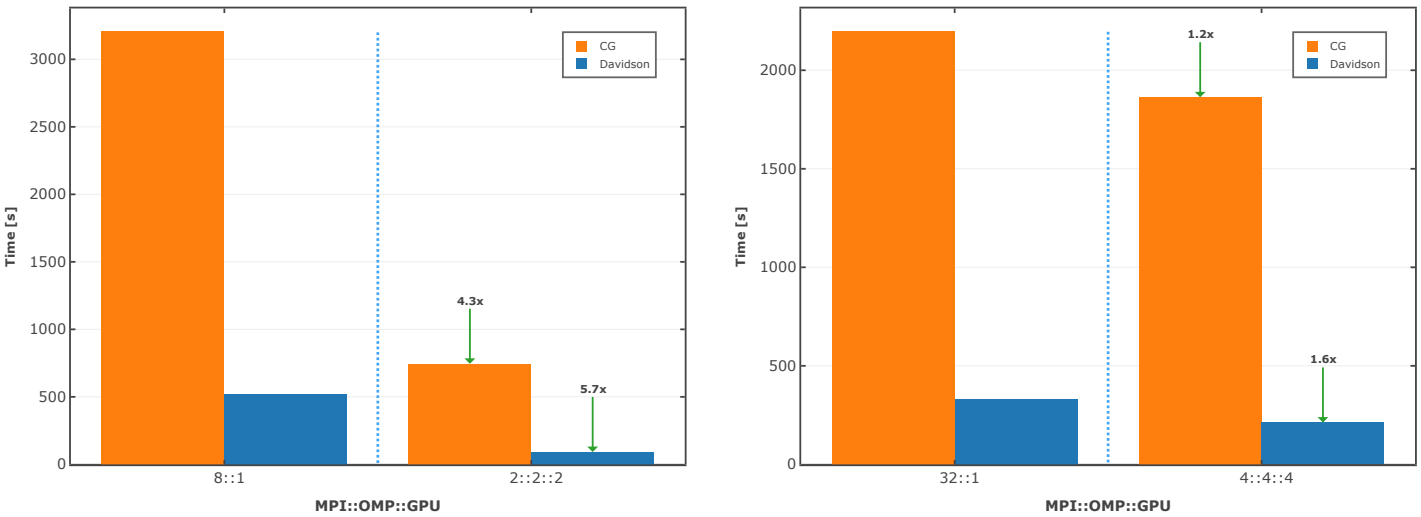


Figure 5.4: Average volatile GPU utilization on Drake with test case AUSURF112

5.2 Comparison between GPU and CPU results

Performance comparison between CPU and GPU versions of the same code are given in the figure 5.5. In all the tested configurations the GPU systems are outperforming the CPU ones. Adding pool parallelism (N_K pools) can improve the performance of a fixed number of CPU or GPU with an increase of N_K. This behaviour is attributed to 3D FFT computations and the eigensolver [7]. The time to solution similarity between one node of Marconi with BWD processors and one node on DAVIDE suggest that the GPU implementation of QE is very competitive to the CPU one. This can be imputed to efficient implementation of FFT 3D library and the custom GPU eigensolver which delivers the performance similar to that of ScaLAPACK and ELPA.



(a) (b)

Figure 5.5: performance comparison of CPU and GPU for the test case of AUSURF112 (a) 8-core Intel Xeon E5-2620 v4 @ 2.10GHz with 2 Quadro P100, (b) 1 node (32 processors) BWD (Marconi) with 1 GPU node (DAVIDE)

CHAPTER 6

Future Work

In ESLW project, the current implementation of PPCG only includes the Γ-point calculation. This has to be extended to general k-point calculations. The ultimate aim of this project is to develop a self-sustained Kohn–Sham solver module with many different choices for iterative diagonalization methods. With this in mind, alternative solvers such as ParO has to be implemented in the library. An interesting extension to the porting of QE to GPU would include a GPU version of the PPCG solver. There are places in the GPU code of the CG solver that can be improved further. Lots of places where loops can be combined to reduce the number of distinct CUF kernels to save memory bandwidth and reduce latency losses. An immediate step in the porting process is the Γ-point calculation. This part of the CG method is currently under development, and it's near completion. Other portions for porting include, extending the code to non-collinear cases, exact exchange calculations, etc.,

CHAPTER 7

Conclusion

Distinct approaches which make Quantum ESPRESSO suitable to be executed on massively parallel supercomputers have been developed and tested in this work. We have used libraries and programming models which make use of the cutting edge hardware technologies such as GPU's. By making the most expensive parts of the QE code such as iterative diagonalization efficient one would be able to study large quantum systems and push the boundaries of the material research. We checked the validity and accuracy of PPCG solution and GPU version of the CG with the known test cases and benchmarked their performance. GPU version of the code produces accurate results with a time to solution which is multiple factors lower than a reference CPU system. From the initial impression, the same can be said about the PPCG solver. For the version of QE ported to GPU systems a trend of increase in performance can be observed from Tesla K80 GPU (Kepler) to Tesla P100 GPU (Pascal) cards due to the faster memory subsystem and higher double precision performance. This trend can be expected to follow in the upcoming generation of NVIDIA GPUs. The work presented here utilises the current advancements in the software and hardware technology which constitutes an instance of an HPC approach which is most useful towards scientific computing.

PPCG Solver implemented in the ESLW is available on Gitlab at: `https://gitlab.e-cam2020.eu/esl/ESLW_Drivers`

CUDA-accelerated version of QE is open-source and available on Github at: `https://github.com/fspiga/qe-gpu`

Bibliography

[1] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials," *Journal of Physics: Condensed Matter*, vol. 21, no. 39, p. 395502, sep 2009. [Online]. Available: https://doi.org/10.1088%2F0953-8984%2F21%2F39%2F395502

[2] R. M. Dreizler and E. K. U. Gross, *Density Functional Theory*. Springer Berlin Heidelberg, 1990. [Online]. Available: https://doi.org/10.1007%2F978-3-642-86105-5

[3] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. D. Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. O. de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, "Advanced capabilities for materials modelling with quantum ESPRESSO," *Journal of Physics: Condensed Matter*, vol. 29, no. 46, p. 465901, oct 2017. [Online]. Available: https://doi.org/10.1088%2F1361-648x%2Faa8f79

[4] [Online]. Available: http://www.gnu.org/licenses/

[5] F. Spiga and I. Girotto, "phiGEMM: A CPU-GPU library for porting quantum ESPRESSO on hybrid systems," in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, feb 2012. [Online]. Available: https://doi.org/10.1109%2Fpdp.2012.72

[6] [Online]. Available: https://github.com/NVIDIA/Eigensolver_gpu

[7] J. Romero., E. Phillips., M. Fatica., F. Spiga., and P. Giannozzi., "A performance study of quantum espresso's pwscf code on multi-core and gpu systems, 8th ieee international

workshop on performance modeling, benchmarking and simulation of high performance computer systems (pmbs17)," *Lecture Notes in Computer Science, Springer, Denver*, 2017. [Online]. Available: http://doi.org/10.5281/zenodo.823200

[8] [Online]. Available: https://gitlab.e-cam2020.eu/esl/ESLW_Drivers

[9] [Online]. Available: http://www.netlib.org/lapack/lawnspdf/lawn99.pdf

[10] E. Vecharynski, C. Yang, and J. E. Pask, "A projected preconditioned conjugate gradient algorithm for computing many extreme eigenpairs of a hermitian matrix," *Journal of Computational Physics*, vol. 290, pp. 73–89, jun 2015. [Online]. Available: https://doi.org/10.1016%2Fj.jcp.2015.02.030

[11] P. Hohenberg and W. Kohn, "Inhomogeneous electron gas," *Physical Review*, vol. 136, no. 3B, pp. B864–B871, nov 1964. [Online]. Available: https://doi.org/10.1103%2Fphysrev.136.b864

[12] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," *Physical Review*, vol. 140, no. 4A, pp. A1133–A1138, nov 1965. [Online]. Available: https://doi.org/10.1103%2Fphysrev.140.a1133

[13] J. P. Perdew, J. A. Chevary, S. H. Vosko, K. A. Jackson, M. R. Pederson, D. J. Singh, and C. Fiolhais, "Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation," *Physical Review B*, vol. 46, no. 11, pp. 6671–6687, sep 1992. [Online]. Available: https://doi.org/10.1103%2Fphysrevb.46.6671

[14] J. Tao, J. P. Perdew, V. N. Staroverov, and G. E. Scuseria, "Climbing the density functional ladder: Nonempirical meta–generalized gradient approximation designed for molecules and solids," *Physical Review Letters*, vol. 91, no. 14, sep 2003. [Online]. Available: https://doi.org/10.1103%2Fphysrevlett.91.146401

[15] S. R. Billeter, A. J. Turner, and W. Thiel, "Linear scaling geometry optimisation and transition state search in hybrid delocalised internal coordinates," *Physical Chemistry Chemical Physics*, vol. 2, no. 10, pp. 2177–2186, 2000. [Online]. Available: https://doi.org/10.1039%2Fa909486e

[16] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, feb 2005. [Online]. Available: https://doi.org/10.1109%2Fjproc.2004.840301

[17] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, jan 1999. [Online]. Available: https://doi.org/10.1137%2F1.9780898719604

[18] L. S. Blackford, S. Hammarling, G. Henry, D. Walker, J. Choi, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, K. Stanley, and J. Dongarra, "ScaLAPACK," in *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing '96*. ACM Press, 1996. [Online]. Available: https://doi.org/10.1145%2F369028.369038

[19] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, "The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science," *Journal of Physics: Condensed Matter*, vol. 26, no. 21, p. 213201, may 2014. [Online]. Available: https://doi.org/10.1088%2F0953-8984%2F26%2F21%2F213201

[20] T. A. Arias, M. C. Payne, and J. D. Joannopoulos, "Ab initiomolecular-dynamics techniques extended to large-length-scale systems," *Physical Review B*, vol. 45, no. 4, pp. 1538–1549, jan 1992. [Online]. Available: https://doi.org/10.1103%2Fphysrevb.45.1538

[21] A. Qteish, "Conjugate-gradient methods for metallic systems and band-structure calculations," *Physical Review B*, vol. 52, no. 20, pp. 14 497–14 504, nov 1995. [Online]. Available: https://doi.org/10.1103%2Fphysrevb.52.14497

[22] Y. Pan, X. Dai, S. de Gironcoli, X.-G. Gong, G.-M. Rignanese, and A. Zhou, "A parallel orbital-updating based plane-wave basis method for electronic structure calculations," *Journal of Computational Physics*, vol. 348, pp. 482–492, nov 2017. [Online]. Available: https://doi.org/10.1016%2Fj.jcp.2017.07.033

[23] [Online]. Available: https://www.e-cam2020.eu

[24] [Online]. Available: https://github.com/fspiga/qe-gpu-plugin